FlashStieltjes:
Fast and Memory-Efficient Flash-Style Attention with Algebraic Normalization

Alex Gulko

The Ohio State University

gulko.5@osu.edu

An Undergraduate Honors Research Thesis

Presented in Partial Fulfillment of the Requirements for Graduation with
Honors Research Distinction in Computer Science and Engineering
at The Ohio State University

April 2026

Advisor: Sachin Kumar
Committee Member: Arnab Nandi

Abstract

Softmax is the default attention normalization in Transformers, but its exponential form is only one of many possible maps to the probability simplex. We study *Stieltjes attention*, which replaces softmax with the algebraic weights $p_j = (\lambda - s_j)^{-q}$, where a per-row pole λ enforces $\sum_j p_j = 1$. We present *FlashStieltjes*, an open-source flash-style Triton forward kernel for Stieltjes attention that delivers a $3.37\times$ geomean forward speedup on NVIDIA A100 and $3.78\times$ on H100 over a PyTorch Stieltjes reference, along with a $4.9\times$ reduction in peak memory. Using this kernel, we train nanoGPT-scale Transformers on a suite of algorithmic-reasoning tasks and find that Stieltjes is a drop-in alternative to softmax: accuracy matches the softmax baseline within ≤ 0.03 across the suite, and at moderate q the model extrapolates to $64\times$ its training length on a subtle-needle retrieval task. Source code is released under the MIT License at `https://github.com/gulkoa/stieltjes-triton`.

Contents

1 Introduction 3

2 Background and Related Work 4

3 Method 8

4 Experiments 11

5 Discussion 17

6 Conclusion 19

A Implementation details 21

B One-head one-layer characterization 22

Acknowledgments

I gratefully acknowledge the Ohio Supercomputer Center [Ohio Supercomputer Center, 1987] for providing the computational resources used in this work: the Ascend cluster (NVIDIA A100-PCIE-40 GB) and the Cardinal cluster (NVIDIA H100-SXM5-80 GB). The Triton throughput benchmarks, nanoGPT training, and length-extrapolation evaluations reported here were all run on OSC infrastructure.

I am grateful to Jack Taylor, Tarun Kathuria, and my advisor Sachin Kumar for their collaboration on the broader Stieltjes-transform simplex-map framework from which this thesis emerged.

This thesis presents one fragment of that larger ongoing research project. The Triton implementation and benchmarking reported here are my own contribution; the underlying simplex-map framework is collaborative, and full-scope results and co-authored work will be published separately.

1 Introduction

The softmax function is the nearly-universal normalization in Transformer attention mechanisms [Vaswani et al., 2017]. Given query–key scores $s_j = \mathbf{q}^\top \mathbf{k}_j / \sqrt{d}$, standard attention computes

$$\mathbf{o} = \sum_{j=1}^{n} \frac{\exp(s_j)}{\sum_{k=1}^{n} \exp(s_k)} \mathbf{v}_j \,. \tag{1}$$

Softmax is easy to compute, differentiate, and optimize, which gave it widespread adoption among modern state-of-the-art models. However, any function that maps a real-valued score vector to the probability simplex Δ^{n-1} can serve as an attention normalization, and the exponential is only one such mapping.

Softmax's exponential form has a known out-of-distribution dispersion property. Veličković et al. [2025] proved that, under specific assumptions, softmax attention coefficients converge toward the uniform distribution at sufficiently long sequence lengths out-of-distribution: because the logit spread $\delta = \max_j s_j - \min_j s_j$ is bounded by the compactness of the input space, the top attention weight $\exp(\delta)/n$ eventually vanishes as $n \to \infty$.

Scope and assumptions. This dispersion result is a *theoretical bound* that holds under several simplifying constraints: (i) a finite input vocabulary, which implies bounded embeddings and therefore bounded logits; (ii) analysis of a single attention head in isolation, without residual connections or layer normalization; and (iii) evaluation at sequence lengths well beyond the training distribution. In practice, mechanisms such as learned temperature scaling, residual-stream mixing, and multi-head interaction can modulate the effective sharpness of attention and may slow or partially counteract dispersion in deployed models. We invoke the result here not as a claim about universal Transformer behaviour, but as a precise characterisation of the softmax *map itself*: for any fixed logit range, the exponential kernel's normalisation constant grows linearly in n, diluting individual weights at a rate set by the functional form.

We observe this dilution directly in our one-layer experiments (Section 4.5): attention entropy grows as $\log(N_{\text{eval}})$, and the softmax weight profile visibly flattens at longer evaluation lengths. That the rate of dilution is intrinsic to the exponential form motivates exploring simplex maps with different tail behaviour.

A growing body of work has explored alternatives. Sparsemax [Martins and Astudillo, 2016] projects scores onto the simplex and produces genuinely sparse outputs. The α-entmax family [Peters et al., 2019, Correia et al., 2019] generalizes both softmax ($\alpha = 1$) and sparsemax ($\alpha = 2$) through Tsallis entropy [Tsallis, 1988]. Linear attention [Katharopoulos et al., 2020] removes the normalization entirely by approximating the softmax kernel. These options have remained largely theoretical or restricted to bespoke, non-flash-style implementations, limiting practical evaluation at modern Transformer scales.

Within this space, the Stieltjes transform offers a particularly natural alternative. Classically, the Stieltjes transform of a positive measure μ on $\mathbb{R}$ is the resolvent $G(z) = \int (z - t)^{-1} d\mu(t)$, a cornerstone of random-matrix theory and spectral analysis. For the discrete score measure $\sum_j \delta_{s_j}$ and a pole λ above the maximum, this becomes $G(\lambda) = \sum_j (\lambda - s_j)^{-1}$; dividing each term by $G(\lambda)$ yields a probability vector over keys. Raising the exponent to general $q > 0$ extends this to the family $(\lambda - s_j)^{-q}$. This replaces softmax's exponential kernel with an *algebraic* kernel whose order parameter q directly controls concentration: for any fixed λ, larger q sharpens the weight assigned to the top-scoring keys while preserving polynomial—rather than exponential—decay in the tail. Stieltjes attention therefore exposes the concentration–dispersion tradeoff as a single continuous hyperparameter, rather than burying it in the choice of temperature or truncation threshold.

In this work we present *FlashStieltjes*, a flash-style Triton kernel for *Stieltjes attention*. The attention weights take the algebraic form

$$p_j = (\lambda - s_j)^{-q} \,, \qquad \text{where } \lambda \text{ satisfies } \sum_{j=1}^{n} (\lambda - s_j)^{-q} = 1 \,. \tag{2}$$

The order parameter $q > 0$ alters the tail behaviour of the attention distribution: $q = 1$ yields harmonic normalization, and higher q produces sharper peaks with heavier polynomial tails than

softmax's exponential decay. Finding λ for each query row is the central computational challenge, which we address with a row-wise Newton–Raphson solver.

Stieltjes attention does not admit an online flash decomposition: the pole λ depends globally on all scores in a row, so no single-pass tiled algorithm exists. To our knowledge, this is the first flash-style tiled implementation of a dense algebraic attention normalization requiring iterative rootfinding. The closest prior work, AdaSplash [Gonçalves et al., 2025], implements a tiled Halley-bisection kernel for sparse α-entmax and exploits the resulting sparsity pattern, whereas Stieltjes attention produces dense weights and relies on the rapid Newton–Raphson convergence of the pole-finding equation (2–3 iterations at low q).

1.1 Contributions

1.1.1 FlashStieltjes: a multi-pass flash-style forward kernel

We introduce a multi-pass tiling scheme that preserves the $O(Nd)$ memory footprint of FlashAttention despite Stieltjes's global pole coupling: one row-wise max sweep, T Newton–Raphson sweeps, and one output-accumulation sweep over $(\mathbf{K}, \mathbf{V})$ tiles. The *forward* kernel implementing this scheme is the main kernel artifact of this work (Section 3), recomputing scores on-the-fly from the saved pole λ_i and auxiliary sum D_i. Backward kernels for $\nabla\mathbf{Q}$, $\nabla\mathbf{K}$, and $\nabla\mathbf{V}$ pass random-input correctness tests but are numerically unstable on trained attention patterns (Section 3.3.2); a stable Triton backward is left to future work, and all accuracy-producing training in this paper uses a PyTorch-reference Stieltjes path. Source at `https://github.com/gulkoa/stieltjes-triton`.

1.1.2 Throughput and memory benchmarks on two hardware columns

Forward-kernel speedup and memory reduction against a flash-style PyTorch Stieltjes reference on A100 and H100 (Section 4.2).

1.1.3 Accuracy characterization on algorithmic-reasoning tasks

Softmax-vs-Stieltjes q-sweeps on a suite of CLRS-style tasks across one-layer and six-layer nanoGPT models. In-distribution at six-layer scale, accuracy parity holds across $q \in \{2, 4, 8, 16\}$ under the constant $\lambda_0 = 1.1$ initialization, matching softmax within ≤ 0.03 on all tested tasks; for length extrapolation, a conservative $q \in [2, 4]$ band holds to $64\times$ at one-layer scale on subtle-needle (Sections 4.3–4.5).

1.1.4 The NR iteration count as a training hyperparameter

We document that the Newton–Raphson iteration count T is encoded into trained weights and must match between training and inference—a novel property of Stieltjes attention that we characterize explicitly (Section 3.2).

2 Background and Related Work

2.1 Attention and the Softmax Function

Multi-head attention [Vaswani et al., 2017] is a core part of Transformer architectures. Given queries $\mathbf{Q} \in \mathbb{R}^{N\times d}$, keys $\mathbf{K} \in \mathbb{R}^{N\times d}$, and values $\mathbf{V} \in \mathbb{R}^{N\times d}$, the output is

$$\mathbf{O} = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V}\,, \tag{3}$$

where softmax is applied row-wise. The score matrix $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top/\sqrt{d}$ has shape $N \times N$, so naïve computation requires $O(N^2)$ memory and $O(N^2 d)$ time. For long sequences this quadratic cost dominates both memory and compute: at sequence length $N = 32{,}768$ with head dimension $d = 128$, the score matrix alone consumes 4 GB in float32—more than the model parameters for many architectures.

The softmax function $\mathrm{softmax}(s)_j = \exp(s_j)/\sum_k \exp(s_k)$ maps $\mathbb{R}^n \to \Delta^{n-1}$ (the probability simplex), ensuring non-negative weights that sum to one. This mapping traces its lineage to the

Boltzmann distribution in statistical mechanics and logistic regression in statistics—a historical convention rather than a mathematical necessity. Any function that maps a real-valued score vector to the probability simplex can serve as an attention normalization, and the exponential is only one such mapping.

Three properties of softmax are commonly believed essential for effective attention: (i) non-negativity of weights, (ii) row-wise normalization to sum to one (interpretable as probabilities), and (iii) concentration of mass on high-scoring keys (sparsity-promoting behavior). Recent work has challenged each of these assumptions [Saratchandran et al., 2024, Wortsman et al., 2023, Ramapuram et al., 2024], suggesting that softmax's effectiveness may stem more from implicit regularization of the attention matrix than from these probabilistic properties per se.

2.2 Pathologies of Softmax Attention

Despite its ubiquity, softmax attention exhibits several well-documented failure modes that motivate the search for alternatives.

Attention dispersion at scale. Veličković et al. [2025] proved that softmax attention coefficients must *disperse* – approach the uniform distribution – as the problem size increases beyond the training distribution, for any Transformer with a finite input vocabulary. The core mechanism is that the logit spread $\delta = \max_j s_j - \min_j s_j$ is bounded by the compactness of the input space, so as $n \to \infty$ the softmax output $\exp(\delta/\theta)/n$ vanishes. This fundamental limitation means softmax cannot robustly maintain sharp attention patterns out-of-distribution, degrading performance on tasks requiring precise key retrieval at novel problem sizes. The authors propose adaptive temperature as an ad-hoc mitigation, but the underlying exponential functional form remains the root cause.

Attention sinks. Xiao et al. [2024] identified the *attention sink* phenomenon: in autoregressive Transformers, initial tokens receive disproportionate attention mass regardless of semantic content. A plausible mechanism, following directly from softmax's sum-to-one constraint, is that the model must "dump" excess attention probability somewhere—typically on the globally visible first token. The phenomenon causes performance collapse in streaming settings where early tokens leave the context window. While adding dedicated sink tokens during pretraining mitigates the issue, it represents a workaround for a structural property of the softmax normalization. Alternative normalizations that either relax the sum-to-one constraint [Ramapuram et al., 2024, Zuhri et al., 2025] or change the tail behavior may address the root cause.

Further failure modes. Other documented softmax limitations include rank collapse in deep stacks [Dong et al., 2021] and a length-dependent flattening of the maximum attention weight [Nakanishi, 2025]; these motivate analogous functional-form modifications but are not directly load-bearing for our work.

2.3 Flash Attention

Flash Attention [Dao et al., 2022] eliminates the $O(N^2)$ memory cost by never materializing the full score matrix. Instead, it processes queries and keys in tiles of size $B_M \times B_N$ that fit in GPU SRAM, computing attention outputs incrementally.

This is possible thanks to the *online softmax* [Milakov and Gimelshein, 2018]: the softmax normalization can be decomposed into a running maximum and partial sum of exponentials. When a new tile of keys arrives, previous partial results are rescaled by the ratio of old and new maxima. This allows a single forward pass over the key blocks, maintaining only $O(B_M \cdot d)$ state in registers.

Flash Attention-2 [Dao, 2024] improved parallelism and work partitioning by iterating over query blocks in the outer loop, and Flash Attention-3 [Shah et al., 2024] exploited Hopper-specific features (asynchronous execution via `wgmma`, FP8 tensor cores, and warp specialization) for further speedups. All variants fundamentally rely on the decomposability of $\exp$ under the online softmax algorithm: the normalization constant can be incrementally updated as new key blocks arrive.

The IO-aware analysis underlying Flash Attention models GPU computation through the lens of HBM bandwidth. Following the roofline model [Williams et al., 2009], attention is memory-bound for typical head dimensions ($d = 64$–128), meaning wall-clock time is dominated by HBM reads and

writes rather than arithmetic. Flash Attention minimizes HBM accesses by keeping partial results in SRAM and performing the maximum possible computation per byte loaded.

Stieltjes attention breaks the online softmax decomposition: the normalization constant λ depends on *all* scores simultaneously through a root-finding problem, so no single-pass online algorithm exists. Our multi-pass approach (Section 3.3) trades additional HBM sweeps for the same $O(Nd)$ memory guarantee, while maintaining each query block in SRAM across all passes.

2.4 Alternatives to Softmax

A growing body of work has explored non-softmax attention normalizations, which we organize into four families: sparse simplex projections, pointwise activations, kernel approximations, and differential/structured attention.

2.4.1 Sparse Simplex Projections

Sparsemax [Martins and Astudillo, 2016] replaces softmax with Euclidean projection onto the simplex, producing exactly sparse attention weights where many keys receive exactly zero probability. The α-entmax family [Peters et al., 2019, Correia et al., 2019, Niculae and Blondel, 2017] generalizes this via Tsallis α-entropy [Tsallis, 1988]: $\alpha = 1$ recovers softmax, $\alpha = 1.5$ gives 1.5-entmax, and $\alpha = 2$ gives sparsemax. Blondel et al. [2020] unified these mappings under the Fenchel–Young loss framework, showing that each α-entmax corresponds to the gradient map of a Tsallis entropy regularizer applied to the simplex projection problem.

These sparse normalizations involve solving a thresholding problem (finding which coordinates are nonzero), which is structurally related to our root-finding approach: both require solving an implicit equation to determine the normalization. The key difference is that entmax produces genuinely *sparse* outputs (many exact zeros), while Stieltjes normalization produces *dense* outputs with algebraic (power-law) tails.

Flash-style sparse attention. Gonçalves et al. [2025] recently introduced AdaSplash, a flash-attention-style implementation of α-entmax in Triton. Their approach uses a hybrid Halley-bisection algorithm for the entmax threshold τ, achieving a 7-fold reduction in rootfinding iterations compared to standard bisection. Custom Triton kernels exploit the resulting sparsity by skipping null blocks, approaching—and in some cases surpassing—FlashAttention-2 throughput on long sequences. AdaSplash is the closest existing work to our contribution: both implement flash-style tiled attention with non-softmax normalizations that require iterative rootfinding. The key architectural difference is that AdaSplash leverages *sparsity* for efficiency (skipping zero blocks), while our kernel operates on *dense* algebraic weights and relies on the fast convergence of Newton–Raphson (2–3 iterations vs. 3–5 Halley-bisection iterations for entmax).

2.4.2 Pointwise Activation Functions

An orthogonal line of work replaces softmax with pointwise activations that abandon sum-to-one normalization. Ramapuram et al. [2024] proposed sigmoid attention $\mathbf{O} = \sigma(\mathbf{S}/\sqrt{d} + b)\,\mathbf{V}$, with FlashSigmoid achieving 17% inference speedup over FlashAttention-2 on H100. Wortsman et al. [2023] replaced softmax with $\mathrm{ReLU}(s_j)/N$, showing that sequence-length scaling closes the gap to softmax in vision transformers. Zuhri et al. [2025] proposed Softpick, a rectified variant producing over 95% exact zeros and a 0% attention sink rate while matching softmax performance. These methods all *abandon* the probability simplex (weights are independent per key and do not sum to one); Stieltjes attention instead preserves $\sum_j p_j = 1$ while changing the functional form of the normalization.

2.4.3 Linear and Kernel-Based Attention

Linear attention [Katharopoulos et al., 2020] approximates $\mathrm{softmax}(\mathbf{QK}^\top) \approx \phi(\mathbf{Q})\phi(\mathbf{K})^\top$ for a feature map ϕ, reducing complexity to $O(Nd^2)$ by exploiting the associativity of matrix multiplication. The Performer [Choromanski et al., 2021] refines this with FAVOR+ (Fast Attention Via positive Orthogonal Random features), providing unbiased estimation of the softmax kernel with provably

lower variance. cosFormer [Qin et al., 2022] achieves non-negativity and concentration through a cosine-based re-weighting scheme while maintaining linear complexity.

These methods eliminate the normalization problem entirely but introduce an approximation gap. In particular, linear attention struggles on tasks requiring sharp, sparse attention patterns [Qin et al., 2022], precisely because the kernel decomposition smooths out the peaking behavior of softmax. Stieltjes attention maintains *exact*, non-approximate normalization while changing the functional form, avoiding this approximation gap at the cost of the multi-pass overhead.

Stieltjes attention is orthogonal to sparsity-based efficient-Transformer variants surveyed by Tay et al. [2022]: we change the *normalization function* rather than the *attention pattern or memory strategy*, and our tiling approach can in principle be combined with sparse or sliding-window patterns.

2.4.4 Differential and Structured Attention

The Differential Transformer [Ye et al., 2025] computes attention as the difference of two softmax attention maps, using subtraction to cancel noise and promote sparse attention patterns. This subtraction mechanism is reminiscent of differential amplifiers in electrical engineering. This approach reduces hallucination, improves long-context retrieval, and diminishes activation outliers. Like Stieltjes attention, it modifies the normalization rather than the attention pattern, but does so by combining two softmax operations rather than replacing the exponential function.

2.5 The Stieltjes Transform

The Stieltjes transform of a measure μ is classically defined as $S_\mu(z) = \int (z-t)^{-1}\, d\mu(t)$, appearing in random matrix theory, spectral analysis, and moment problems. In random matrix theory, the Stieltjes transform is the primary analytical tool for deriving limiting spectral distributions: the Marchenko–Pastur law for sample covariance matrices, the semicircle law for Wigner matrices, and their generalizations all follow from fixed-point equations on Stieltjes transforms [Pennington and Worah, 2017]. The transform has been applied to analyze neural networks through the spectral distributions of weight matrices and Jacobians, connecting optimization dynamics to random matrix universality phenomena.

We[1] observed that a discrete, parameterized variant of the transform—$p_j \propto (\lambda - s_j)^{-q}$—defines a valid probability simplex mapping when λ is chosen to enforce normalization. This mapping has several appealing properties for attention:

1. **Purely algebraic.** No exponentials or transcendental functions are required, avoiding the overflow/underflow hazards that necessitate the log-sum-exp trick in softmax.
2. **Power-law tails.** The order parameter q controls tail behavior in a fundamentally different way than softmax temperature. For a score at distance δ from the maximum, softmax assigns weight $\propto e^{-\delta/\tau}$ (exponential decay), while Stieltjes assigns weight $\propto \delta^{-q}$ (algebraic decay). Higher q sharpens the peak but preserves heavier tails than any finite softmax temperature.
3. **Well-conditioned rootfinding.** The normalization constraint $\sum_j (\lambda - s_j)^{-q} = 1$ yields a strictly monotone, convex function of λ, making Newton–Raphson convergence rapid and robust [Nocedal and Wright, 2006].

The algebraic tail behavior is the most distinctive property. In softmax, the ratio of the largest to smallest weight is $\exp(s_{\max} - s_{\min})$, growing exponentially with the logit spread. In Stieltjes normalization, writing $\Delta_j = \lambda - s_j$ for the gap from the pole, the same ratio is $(\Delta_{\max}/\Delta_{\min})^q$—a polynomial rather than exponential gap. This means distant keys retain small but non-negligible attention weight even when the score distribution has large spread, potentially mitigating the attention dispersion and sink phenomena described in Section 2.2.

2.6 Triton and GPU Kernel Programming

Triton [Tillet et al., 2019] is a language and compiler for writing GPU kernels at a higher level of abstraction than CUDA published by OpenAI. Programs operate on blocks of data (tiles), and

[1] Joint work with Jack Taylor and Tarun Kathuria; see the acknowledgments for scope.

Table 1: Comparison of attention normalizations. "Simplex" indicates outputs lie on the probability simplex ($p_j \geq 0$, $\sum_j p_j = 1$). "Tail" describes the asymptotic weight on low-scoring keys. "Online" indicates whether the normalization admits a single-pass algorithm. "Flash impl." indicates whether a memory-efficient GPU kernel exists.

Method	Simplex	Tail	Sparse	Online	Flash impl.
Softmax [Vaswani et al., 2017]	✓	exponential	no	✓	✓
Sparsemax [Martins and Astudillo, 2016]	✓	zero (exact)	✓	—	—
α-entmax [Correia et al., 2019]	✓	zero (exact)	✓	—	✓[†]
Sigmoid [Ramapuram et al., 2024]	—	sigmoid	no	✓	✓
ReLU [Wortsman et al., 2023]	—	zero (ReLU)	✓	✓	—
Polynomial [Saratchandran et al., 2024]	—	polynomial	no	✓	—
Diff. Transformer [Ye et al., 2025]	—	exponential	no	✓	✓
Stieltjes (ours)	✓	**algebraic**	no	—	✓

[†] AdaSplash [Gonçalves et al., 2025], concurrent work (ICML 2025).

the compiler handles thread mapping, memory coalescing, and instruction scheduling. Triton's `@triton.autotune` decorator enables automatic selection among kernel configurations (block sizes, warp counts, pipeline stages) at runtime.

We implement all kernels in Triton rather than CUDA for three reasons: (i) Python-level authoring reduces development time; (ii) autotuning provides hardware-adaptive performance; and (iii) integration with PyTorch's `autograd` is straightforward via `torch.autograd.Function`. Both our work and AdaSplash [Gonçalves et al., 2025] demonstrate that Triton is a viable platform for implementing non-standard attention normalizations with performance competitive with highly optimized CUDA kernels.

2.7 Positioning of This Work

Table 1 summarizes the design space of attention normalizations, positioning Stieltjes attention relative to existing approaches.

Stieltjes attention occupies a unique point in this design space: it is the only normalization that (i) preserves the probability simplex (non-negative weights summing to one), (ii) produces *dense* outputs with algebraic rather than exponential tails, and (iii) has a GPU-efficient flash-style implementation with full training support. The algebraic tail behavior provides a qualitatively different inductive bias from all existing methods: exponential tails (softmax, sigmoid) decay too fast for distant keys, sparse methods (entmax, sparsemax) assign exactly zero weight, and polynomial activations do not enforce any normalization constraint. The trade-off is the multi-pass kernel design required by the global rootfinding problem, which we address through the fast convergence of Newton–Raphson iteration (Section 3.3).

3 Method

3.1 Stieltjes attention

Given queries $\mathbf{Q} \in \mathbb{R}^{N \times d}$, keys $\mathbf{K} \in \mathbb{R}^{N \times d}$, and values $\mathbf{V} \in \mathbb{R}^{N \times d_v}$, let $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top / \sqrt{d}$ be the scaled score matrix. We replace the row-wise softmax in Equation 1 with the *Stieltjes* simplex map:

$$p_{ij}(\lambda_i, q) = \frac{(\lambda_i - s_{ij})^{-q}}{Z_i(\lambda_i, q)}, \qquad Z_i = \sum_{j \leq i} (\lambda_i - s_{ij})^{-q}, \tag{4}$$

where $q > 0$ is an order parameter and λ_i is the row-specific *pole* chosen so that $\sum_j p_{ij} = 1$. By construction $\lambda_i > \max_j s_{ij}$, and p_{ij} decays polynomially in $\lambda_i - s_{ij}$ rather than exponentially, giving Stieltjes attention heavier algebraic tails than softmax.

3.1.1 Newton–Raphson pole finding

Let $F(\lambda_i) = Z_i(\lambda_i, q) - 1$. We solve $F(\lambda_i) = 0$ with the pure Newton update $\lambda_i \leftarrow \lambda_i - F(\lambda_i)/F'(\lambda_i)$. Prior to the solve we center the scores by subtracting the row maximum, $\tilde{s}_{ij} =$

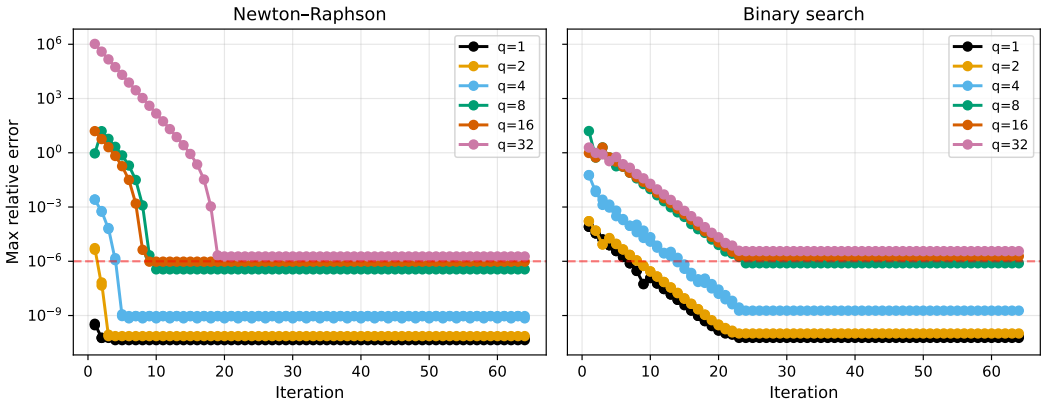


Figure 1: Newton–Raphson (left) vs. binary-search (right) convergence for the pole-finding problem $F(\lambda) = 0$ across $q \in \{1, 2, 4, 8, 16, 32\}$, measured as the maximum residual $|F(\lambda)|$ per iteration on a batch of synthetic score vectors. Newton–Raphson reaches single-precision accuracy within 2–3 iterations for $q \leq 4$, 5–10 for $q \in \{8, 16\}$, and below 20 for $q = 32$; binary search requires 20 or more iterations at every order.

$s_{ij} - \max_k s_{ik}$, so that all $\tilde{s}_{ij} \leq 0$ and the pole of F lies at $\lambda = 0$. Centered this way, a single constant initialization

$$\lambda_i^{(0)} = 1.1 \tag{5}$$

starts every row safely above the pole with a well-conditioned derivative, independent of q, row index, or sequence length. The convergence behaviour on randomly-sampled inputs is shown in Figure 1: Newton–Raphson reaches single-precision accuracy within 2–3 steps at low orders $q \in \{1, 2, 4\}$, 5–10 at $q \in \{8, 16\}$, and below 20 at $q = 32$, whereas a comparable binary-search solver requires $20+$ iterations at every tested order. A per-row initialization of the form $(i+1)^{1/q}$ (the closed-form root for a uniform score distribution) appears natural but in fact *slows* NR at large q by placing $\lambda^{(0)}$ far above the true root, a regime where F' is small and Newton steps overshoot; the constant centered initialization above attains accuracy at high q that the per-row form does not. The initialization choice is load-bearing at training time as well as at inference: six-layer nanoGPT runs under the per-row form converge to a substantially lower ceiling at $q \in \{8, 16\}$. Under Equation 5, six-layer models at $q \in \{8, 16\}$ attain softmax-level or higher accuracy on every tested CLRS task (Section 4.3, Table 2).

3.2 NR iteration count as a training hyperparameter

A fixed iteration count T on Equation 5's initialization produces a converged λ that depends on the input score distribution. Models trained with a specific T therefore encode that choice into their learned QKV projections. Empirically, training with either $T = 3$ or $T = 10$ converges to comparable accuracy on our tasks (Section 4.4), and the two choices produce functionally distinct trained models: evaluation must use the same T that training used. We recommend logging T alongside other attention hyperparameters and treating it as a build-time parameter rather than a runtime toggle. All trained results in this paper fix $T = 3$ for both phases.

3.3 Flash-style Triton kernel

The reference implementation of Equation 4 materializes the full $N \times N$ attention matrix. For $N \geq 4096$ this is the dominant memory term in our nanoGPT training runs and was the motivation for a flash-style fused kernel, following the pattern of FlashAttention [Dao et al., 2022, Dao, 2024].

3.3.1 Forward pass

Our Triton kernel implements the fused forward in three phases that each iterate over K, V tiles while keeping a single query block resident in registers (Figure 2):

1. a row-wise max pass over $\mathbf{QK}^\top$ tiles, stabilizing subsequent arithmetic;

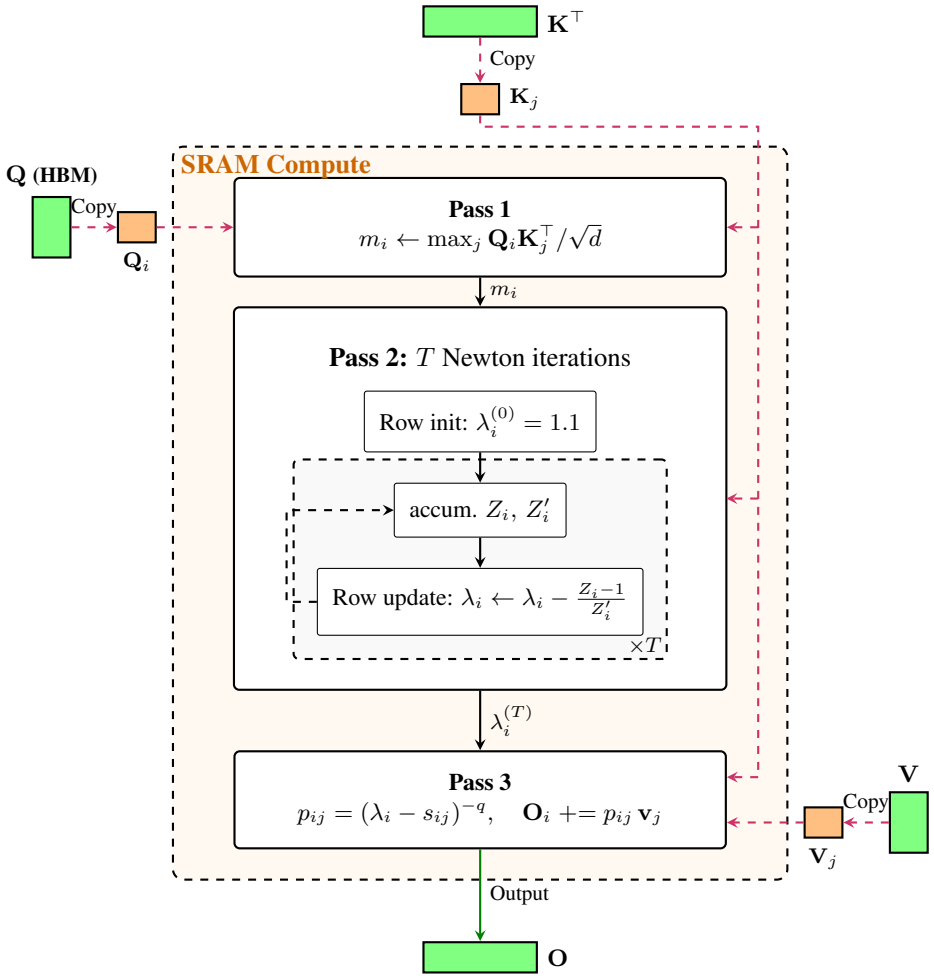


Figure 2: Forward kernel for one query block. $\mathbf{Q}_i$ stays resident in SRAM across three sweeps over $\mathbf{K}$ (and $\mathbf{V}$ on the final sweep); scores s_{ij} are recomputed on the fly each sweep. Pass 2 iterates Newton–Raphson T times (dashed back-edge): each iteration re-sweeps $\mathbf{K}$ to rebuild (Z_i, Z_i') at the current λ_i, then applies a pure Newton update. Only the row-max m_i and converged pole $\lambda_i^{(T)}$ persist between sweeps, keeping peak HBM memory at $O(Nd)$.

2. T Newton iterations; each iteration recomputes tile scores on the fly and accumulates Z_i and $Z_i' = \partial Z_i / \partial \lambda_i$;

3. an output-accumulation pass that recomputes scores once more, forms p_{ij}, and accumulates $\mathbf{PV}$.

The kernel saves, for the backward pass, the converged λ_i and the auxiliary sum $D_i = \sum_{j \le i} (\lambda_i - s_{ij})^{-q-1}$. Peak memory is $O(N \cdot d)$, matching FlashAttention's softmax footprint. At $q = 1$ the power-law evaluation $(\lambda - s_j)^{-q}$ reduces to a single reciprocal plus its square, and the kernel detects this case at compile time to skip the $\exp(-q \log(\cdot))$ pair.

Two further stability devices support the tile-resident recompute path: differences $\Delta_{ij} = \lambda_i - \tilde{s}_{ij}$ are clamped to $\max(\Delta_{ij}, \epsilon)$ (with $\epsilon = 10^{-6}$) to avoid divide-by-zero during early NR iterations, and powers Δ^{-q} are evaluated in log space as $\exp(-q \log \Delta)$ to control intermediate overflow for large q. Causal masking reuses the FlashAttention sentinel convention (-10^{30} on masked keys), which after centering yields $\Delta_{ij} \approx 10^{30}$ and $\Delta_{ij}^{-q} \approx 0$ without an explicit branch in the inner loop. Full autotune configurations and appendix-level details of the backward recompute are deferred to Appendix A.

3.3.2 Backward pass: scope and limitation

We implement three backward kernels (for $\nabla\mathbf{Q}$, $\nabla\mathbf{K}$, and $\nabla\mathbf{V}$) that recompute scores and weights on-the-fly from the saved λ_i and D_i, preserving the $O(N \cdot d)$ memory footprint. These kernels agree with the PyTorch-reference backward to within numerical tolerance on randomly-sampled inputs. However, on the trained attention patterns that arise during nanoGPT optimization, the recompute path is numerically unstable: $\Delta_{ij} = \lambda_i - s_{ij}$ becomes small wherever a row concentrates on a single key, and the resulting Δ_{ij}^{-q-1} factor in the score gradient amplifies low-bit error into gradient norms six to seven orders of magnitude larger than the reference. Consequently, *all accuracy-producing training in this paper uses the PyTorch-reference Stieltjes path*; the Triton forward kernel is used for throughput and memory measurements, and a stable Triton backward remains future work.

3.4 Architecture and training setup

We evaluate Stieltjes attention with nanoGPT-style Transformers at two model scales: a six-layer configuration ($n_{\text{head}} = 6$, $d_{\text{embd}} = 384$, head dimension 64, $\approx 23\,\text{M}$ parameters) used for most CLRS-task training, and a one-layer configuration (same $n_{\text{head}} = 6$ and $d_{\text{embd}} = 384$ width, $\approx 4\,\text{M}$ parameters) used for length-extrapolation probes where saturation within a short training budget is desirable. Both configurations share a byte-level vocabulary of 258 tokens (256 byte values plus `PAD` and `SEPARATOR`), pre-LayerNorm, a $4\times$ GELU MLP, dropout 0.1, and AdamW at learning rate 3×10^{-4} (reduced to 1×10^{-4} at $N \geq 4096$). Attention is either softmax (PyTorch $\mathrm{softmax}(\mathbf{QK}^\top/\sqrt{d})$) or the Stieltjes reference implementation.

For long-context sequences we use an optional `bfloat16` autocast around the forward and loss, halving the $N \times N$ reference-attention footprint. Checkpoints are saved every epoch so jobs walltimed at long sequence length still yield usable models.

3.4.1 Positional encoding and length extrapolation

We report two positional encoding regimes. The default uses the standard GPT-2 learned positional embedding (a table of size $n_{\text{layer}} \times d_{\text{embd}}$ added to token embeddings at each input position). For length-extrapolation evaluations we use *NoPE* ("no positional encoding"), in which the positional embedding table is removed and the model must infer positional information purely from token content and causal masking [Kazemnejad et al., 2023]. NoPE is the necessary regime for evaluating a model at sequence lengths exceeding the maximum position index seen during training: learned positional embeddings degrade immediately outside their trained range, while NoPE models can in principle run at any sequence length.

For length-extrapolation evaluations, a model trained at sequence length N_{train} is evaluated at $N_{\text{eval}} \in \{N_{\text{train}}, 2N_{\text{train}}, 4N_{\text{train}}, \ldots\}$ with the task's `max-arr-len` scaled so that the array portion of the input grows with N_{eval}. Without this scaling, only the padding portion of the input grows and the model produces identical outputs; all length-extrapolation numbers reported below use the scaled protocol.

4 Experiments

We evaluate our Stieltjes attention implementation along three axes: (i) throughput and memory of the Triton forward kernel relative to a PyTorch reference (Section 4.2); (ii) training accuracy of Stieltjes vs. softmax attention at matched model scale on a suite of CLRS-style algorithmic tasks (Section 4.3); and (iii) length-extrapolation behaviour as a function of the Stieltjes order q (Section 4.5), which identifies the q range in which Stieltjes matches softmax on this robustness axis.

4.1 Setup

Throughput benchmarks run on NVIDIA A100-PCIE-40 GB and NVIDIA H100-SXM5-80 GB. All accuracy experiments use the nanoGPT backbone of Section 3.4 at either the six-layer or one-layer configuration. The CLRS tasks are byte-level synthetic problems: *needle* (find a distinctive token in an array), *subtle-needle* (find the $\max+1$ token with margin one above the runner-up), *max* (return the array maximum), *binary_search*, *sorting*, and *bfs*. The CLRS name follows the algorithmic-task style of Cormen, Leiserson, Rivest, and Stein [Cormen et al., 2009]; we do not use the CLRS-30 benchmark

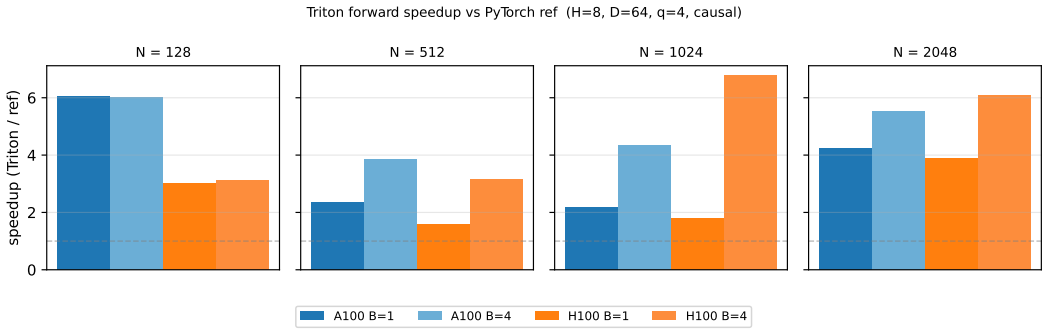


Figure 3: Per-shape forward speedup of the Triton Stieltjes kernel against the PyTorch reference on A100 (blue) and H100 (orange) for representative training shapes: $H = 8$, $D = 64$, $q = 4$, causal, $B \in \{1, 4\}$, $N \in \{128, 512, 1024, 2048\}$. Geometric-mean forward speedup is $3.37\times$ on A100 and $3.78\times$ on H100.

suite. Output-only accuracy is reported on a held-out 2,000-sample validation set. Training uses AdamW with gradient clipping at 1.0 and no learning-rate warmup; the length-extrapolation protocol follows Section 3.4.

4.2 Throughput and memory

We benchmark our Triton forward kernel against two baselines: (i) a flash-style PyTorch reference implementation of the *same Stieltjes normalization* (the primary comparison, which isolates the benefit of tiling over the unfused path), and (ii) `torch.nn.functional.scaled_dot_product_attention` (SDPA, PyTorch's built-in scaled dot-product attention; under our dtype/shape conditions it dispatches to a FlashAttention backend, yielding the state-of-the-art softmax kernel) as a throughput ceiling. All benchmarks use batch $B \in \{1, 4\}$, heads $H = 8$, sequence lengths $N \in \{128, 512, 1024, 2048\}$, head dimension $D = 64$, $q = 4$, causal masking.

Against the PyTorch Stieltjes reference, the geometric-mean forward speedup is $3.37\times$ on A100 and $3.78\times$ on H100. Figure 3 shows the per-shape paired bars. Against FlashAttention softmax, our kernel is slower: at the representative training shape $(B, H, N, D) = (4, 8, 2048, 64)$ (causal), our kernel reaches 18.9 TFLOPS while `scaled_dot_product_attention` reaches 171.4 TFLOPS on the same hardware. This throughput gap is structural, not an optimization artifact, and is analyzed below.

4.2.1 Absolute TFLOPS and peak utilization

Relative speedup is informative but hardware-dependent; absolute throughput and fraction of peak tensor-core throughput tell the rest of the story. Figure 4 reports forward TFLOPS for the Triton kernel and the PyTorch reference across $N \in \{128, \ldots, 4096\}$, $B \in \{1, 4\}$, $D \in \{64, 128\}$, $q \in \{2, 4\}$, causal and non-causal, on A100 and H100. We follow the FlashAttention convention of $2BHN^2D$ floating-point operations for causal forward and $4BHN^2D$ for non-causal. The right-column panels report the fraction of each GPU's peak bf16 tensor-core throughput (A100 = 312 TFLOPS, H100 = 989 TFLOPS). Here B, H, N, D are batch size, number of heads, sequence length, and per-head dimension, respectively; the count reflects the two leading matmuls ($QK^\top$ and the weighted sum with V), each contributing $2BHN^2D$ multiply–add operations over the full grid. Causal masking halves both matmuls to the lower triangle. The normalization step itself (softmax or Stieltjes) is $O(N^2)$ elementwise work and is not counted.

At the training shape $(B, H, N, D) = (4, 8, 2048, 64)$ with $q = 4$ and causal masking, the Triton kernel allocates 329 MB of peak CUDA memory against 1609 MB for the PyTorch reference—a $4.9\times$ reduction at this shape. The advantage compounds at longer contexts: at $N = 16{,}384$ (batch size 1), the Triton kernel uses 65 MB versus 21 GB for the unfused reference—a $320\times$ reduction. Section 4.2.2 and Figure 5 characterise the full scaling behaviour to $N = 131{,}072$.

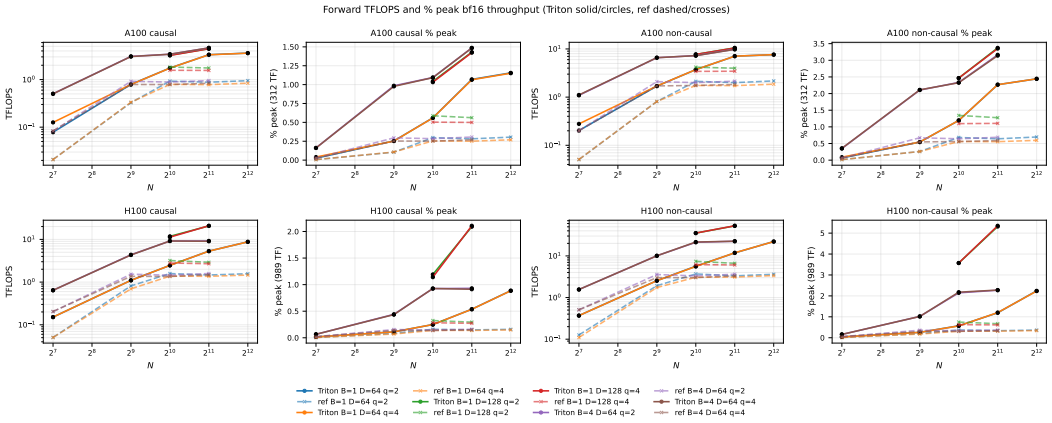


Figure 4: Forward TFLOPS and fraction of peak bf16 tensor-core throughput for the Triton Stieltjes kernel (solid, circles) versus the PyTorch reference (dashed, crosses), across all benchmark shapes. Top row: A100; bottom row: H100. Within each row the four columns are causal absolute TFLOPS, causal % peak, non-causal absolute TFLOPS, and non-causal % peak. Triton reaches a larger fraction of peak throughput than the reference at every shape, with the gap widening as N grows.

Why Stieltjes cannot match FlashAttention throughput. FlashAttention needs two HBM passes per forward call; Stieltjes needs $T + 2 = 5$ at our default $T = 3$ (one for row maxima, T Newton–Raphson sweeps, one for weight accumulation), a $2.5\times$ memory-bandwidth overhead independent of the arithmetic cost of $(\lambda - s)^{-q}$ versus $\exp(s)$. At $(B, H, N, D) = (4, 8, 2048, 64)$ causal on H100, this manifests as 18.9 TFLOPS versus FlashAttention's 171.4 TFLOPS —a $9\times$ gap consistent with the pass-count ratio plus a small arithmetic-intensity penalty. Mitigations (fusing NR iterations, warm-starting λ_0 from adjacent queries, or a two-pass algebraic approximation) are deferred to future work.

4.2.2 Long-context scaling

Figure 5 extends the comparison to $N \in \{1{,}024, \ldots, 131{,}072\}$ at batch size $B = 1$, heads $H = 8$, head dimension $D = 64$, causal, $q = 4$. We include all four implementations: Triton Stieltjes, FlashAttention (`scaled_dot_product_attention`), the unfused PyTorch Stieltjes reference, and naive materialised softmax.

Two observations follow. First, the $O(Nd)$ memory property is confirmed: Triton Stieltjes memory doubles with each $2\times$ increase in N (4 MB at $N = 1{,}024$ to 520 MB at $N = 131{,}072$), matching FlashAttention within $1\,\%$ at every point. The unfused implementations follow the $O(N^2)$ theory line exactly and exhaust the 40 GB A100 budget between $N = 16{,}384$ and $N = 32{,}768$.

Second, the throughput gap between Triton Stieltjes and FlashAttention is stable across scale: 3.5–3.7 % of FlashAttention TFLOPS from $N = 2{,}048$ to $N = 131{,}072$. Both kernels improve with N as tile occupancy grows, but at the same rate. The relative cost of Stieltjes attention is therefore a fixed overhead, not one that grows with context length.

4.3 Accuracy on CLRS tasks

Across the CLRS suite at training sequence length $N = 128$, we train matched softmax and Stieltjes models (six-layer, 30 epochs on $50{,}000$ samples, batch size 64) with the NR-3 Stieltjes variant. Table 2 reports validation accuracy at convergence.

At the task-dependent optimal order $q \in \{2, 4\}$, Stieltjes matches softmax within 0.005 accuracy on every task. Accuracy parity at the six-layer / $N \leq 2048$ regime is therefore well-established; the informative behaviour emerges at longer evaluation context, which we characterize in Section 4.5.

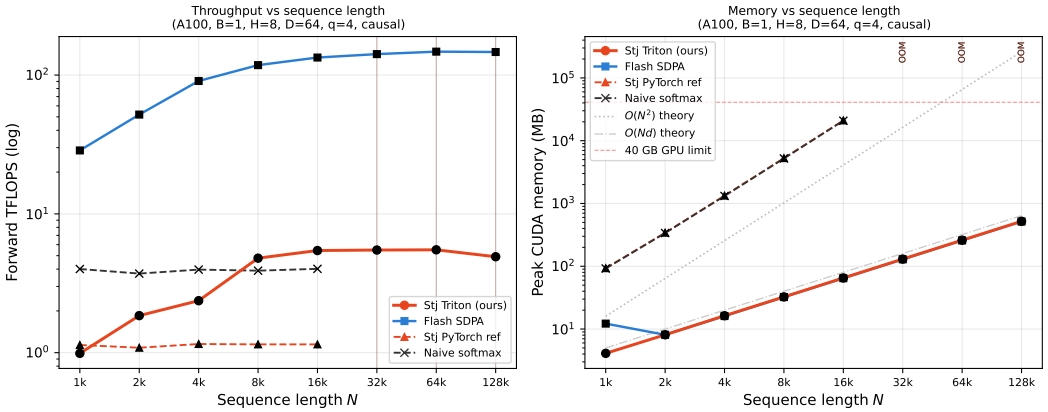


Figure 5: Long-context scaling on A100-40 GB. *Left*: forward TFLOPS vs N (log–log). *Right*: peak CUDA memory vs N (log scale) with theoretical $O(N^2)$ and $O(Nd)$ lines and the 40 GB hardware limit. Both tiled kernels (Triton Stieltjes and Flash SDPA) maintain $O(Nd)$ memory and run to $N = 131{,}072$; both unfused implementations OOM at $N = 32{,}768$. At $N = 16{,}384$ the tiled kernels use ≈ 65 MB while the unfused implementations require ≈ 21 GB — a $\mathbf{320}\times$ reduction. On throughput, Triton Stieltjes outperforms the PyTorch Stieltjes reference by $\approx 5\times$ at mid-range N (visible on the log scale); Flash SDPA remains a structural ceiling above both.

Task	softmax	stj q=1	stj q=2	stj q=4	stj q=8	stj q=16	stj q=32	stj q=64
binary_search	0.999	0.982	0.998	1.000	$0.953^{\star}_{\pm.066}$	$0.978^{\star}_{\pm.017}$	$\leq .05^{\dagger}$	$\leq .05^{\dagger}$
max	0.997	0.996	0.996	0.994	$0.995^{\star}_{\pm.000}$	$0.960^{\star}_{\pm.002}$	$0.171^{\dagger}$	$0.304^{\dagger}$
bfs	$0.995^{\star}_{\pm.000}$	$0.990^{\star}_{\pm.002}$	$0.994^{\star}_{\pm.000}$	$0.993^{\star}_{\pm.000}$	$0.989^{\star}_{\pm.003}$	$0.102^{\dagger}$	$0.145^{\dagger}$	$0.119^{\dagger}$
sorting	0.998	0.997	0.997	0.998	$0.975^{\star}_{\pm.015}$	—	—	—

Table 2: Stieltjes matches softmax within ≤ 0.03 on every tested task. Validation accuracy at convergence on CLRS tasks, six-layer nanoGPT, $N = 128$, training samples 50,000, batch size 64, Stieltjes rows 30–50 epochs and softmax rows 50 epochs, scored under the `accuracy_fixed` metric. At task-dependent optimal $q \in \{2, 4\}$, Stieltjes is within ≤ 0.005 of softmax on every task; on binary_search $q = 4$ exceeds softmax by 0.001. On *bfs*, Stieltjes at all $q \in \{1, 2, 4, 8\}$ matches softmax within ≤ 0.006 (uniform parity across the sweep with no ceiling break); on *binary_search*, Stieltjes at $q \in \{8, 16\}$ stays within ≤ 0.03 of softmax. Entries marked * are reported as $\text{mean}_{\pm\sigma}$ over two random seeds (seed 1 on H100 plus seed 42 on A100; identical training config, separate hardware); these cells use the constant $\lambda_0 = 1.1$ initialization (Section 3.1.1). *binary_search* $q = 8$ shows the largest inter-seed gap (0.906/1.000, $\sigma = 0.066$); the remaining cells agree across seeds to within $\sigma \leq 0.017$. On *sorting*, all reported cells (softmax and Stieltjes $q \in \{1, 2, 4, 8\}$) cluster in $[0.965, 0.998]$, so Stieltjes matches softmax across the tested q range. Values marked † use a per-row λ_0 initialization, retained for $q \in \{32, 64\}$ where the high-q regime is unstable regardless of initialization. Checkpoint-based `accuracy_fixed` evaluation at epoch 30 on 5000 held-out samples agrees with the reported val_accuracy to within ± 0.002.

4.4 NR iteration count as a training hyperparameter

Section 3.2 argued that the Newton iteration count T is encoded into the trained attention. We verify this empirically.

Training with $T = 3$ and $T = 10$ both converge to saturation (≥ 0.99 validation accuracy) within the same epoch count on binary_search at $q = 4$ and on subtle-needle at $q = 4$; the training dynamics differ only by mid-training loss shape. The two trained models are functionally distinct at inference: a binary_search model trained at $T = 3$ and evaluated at $T = 10$ produces accuracy 0.14, versus 0.9996 when T is matched. The same holds in reverse (train $T = 10$, eval $T = 3$). T must therefore be reported and fixed at inference time; we fix $T = 3$ for all results below.

Model	$N = 128$	512	2,048	4,096	8,192	16,384	32,768	131,072
softmax	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
stj $q = 4$	1.000	1.000	1.000	1.000	1.000	—	—	—
stj $q = 8$	0.901	0.901	0.901	0.901	0.901	—	—	—
stj $q = 16$	0.873	0.873	0.873	0.873	0.873	—	—	—

Table 3: Subtle-needle validation accuracy at one-layer scale with NoPE, training at $N_{\text{train}} = 128$ with per-sample needle value uniformly in $[20, 126]$, evaluating at the listed N_{eval} with fixed `max-arr-len` $= 120$. All Stieltjes rows use the PyTorch reference backend with $T = 3$ Newton–Raphson iterations and constant $\lambda_0 = 1.1$ initialization (Section 3.1.1), matched to training; the softmax row uses `torch.nn.functional.scaled_dot_product_attention` (flash), which permits extrapolation to $N_{\text{eval}} = 131{,}072$ ($1024 \times N_{\text{train}}$). Stieltjes at $q = 4$ matches softmax at 1.000 across all tested lengths; higher q shows successively lower training accuracy (0.901 at $q = 8$, 0.873 at $q = 16$) that remains *constant* across all evaluation lengths—a lower ceiling, not extrapolation failure. Stieltjes rows at $N_{\text{eval}} \geq 16{,}384$ are omitted because the PyTorch reference backend is out-of-memory at those sequences; the Triton forward kernel would fit but exhibits a not-yet-root-caused accuracy dip at $N_{\text{eval}} \in \{2048, 4096\}$ that recovers at 8192, so for consistency all Stieltjes accuracy rows above use the reference backend.

Model (six-layer, $N_{\text{train}} = 2048$)	$1\times$	$2\times$	$4\times$	$8\times$
softmax	1.000	1.000	1.000	1.000
stj $q = 4$	0.916	0.672	0.112	0.008

Table 4: Six-layer needle (distinctive margin) with NoPE at $N_{\text{train}} = 2048$, evaluated at $\{2048, 4096, 8192, 16384\}$. At the larger six-layer scale, Stieltjes at $q = 4$ begins to degrade outside the training length while softmax remains saturated. Attention-output standard deviation measurements corroborate the mechanism: the Stieltjes block's output standard deviation decays 35–50 % at $8\times$, concentrated in the deeper layers; the softmax block's output is stable across N_{eval}.

4.5 The q–scale extrapolation profile

We train Stieltjes and softmax with NoPE at sequence length N_{train} and evaluate at $N_{\text{eval}} \in \{N_{\text{train}}, 2N_{\text{train}}, 4N_{\text{train}}, 8N_{\text{train}}\}$ (and beyond) using the scaled `max-arr-len` protocol of Section 3.4. We probe two regimes: a one-layer model on subtle-needle at $N_{\text{train}} = 128$ that admits $64\times$ extrapolation (Table 3), and a six-layer model on needle at $N_{\text{train}} = 2048$ that probes the large-model regime (Table 4).

Two observations follow. First, the dispersion mechanism predicted by Veličković et al. [Veličković et al., 2025] is directly visible in our measurements: attention entropy grows as $\log(N_{\text{eval}})$ (Figure 7) and the softmax attention weight profile flattens at longer evaluation lengths (Figure 6). Stieltjes at higher q shifts both curves toward sharper attention, quantifying the mechanism softmax alternatives are designed to address. Task accuracy on subtle-needle, however, remains at 1.000 for softmax out to $N_{\text{eval}} = 131{,}072$: at $N_{\text{train}} = 128$ the task does not require sharp attention to solve correctly even as the underlying weight distribution flattens, which is why Veličković et al.'s reported task failure uses a tighter $N_{\text{train}} = 16$ probe. At six-layer depth and $N_{\text{train}} = 2048$ the dispersion begins to cost task accuracy (Table 4).

Second, within the Stieltjes family, at $q = 4$ the trained model matches softmax at 1.000 across every tested evaluation length. Higher q shows successively lower training accuracy: 0.901 at $q = 8$ and 0.873 at $q = 16$; crucially, these values are *constant* from $N_{\text{eval}} = 128$ to $8{,}192$, so the lower ceiling is a training-quality effect, not an extrapolation failure. The $q \in [2, 4]$ band is safe at all model scales tested; at one-layer scale, $q = 8$ is also usable.

Figure 6 visualizes the dispersion mechanism directly. Each row shows the attention weight profile (top-32 weights per query, sorted by magnitude) for one model variant, across evaluation sequence lengths $N_{\text{eval}} \in \{128, 512, 2048\}$, for the one-layer subtle-needle task trained at $N_{\text{train}} = 128$. At the training length, softmax and Stieltjes at $q = 4$ both assign moderate concentration to a handful of keys. As N_{eval} grows, softmax attention disperses: weights flatten and the bars in the heatmap



Figure 6: Attention weight patterns for the one-layer subtle-needle task (trained at $N_{\text{train}} = 128$), showing the top-32 weights of the last query at each evaluation length. Rows: softmax and Stieltjes at $q \in \{4, 8, 16, 24, 32\}$. Columns: $N_{\text{eval}} \in \{128, 512, 2048\}$. Softmax and $q = 4$ attention disperses as sequence length grows; higher-q variants maintain sharper patterns but at the cost of lower training accuracy (Table 3). $N_{\text{eval}} = 8192$ is omitted because the PyTorch reference backend OOMs at that length across all six rows.

fade. Stieltjes at higher q concentrates more sharply at training length but trades off against the lower training accuracy visible in Table 3.

Figure 7 quantifies the same effect. Mean attention entropy grows as $\log(N_{\text{eval}})$ for all variants (matching the maximum-entropy dotted reference), but Stieltjes at higher q shifts the curve downward: the model remains more concentrated throughout, consistent with polynomial tails dispersing more slowly than the exponential tails of softmax.

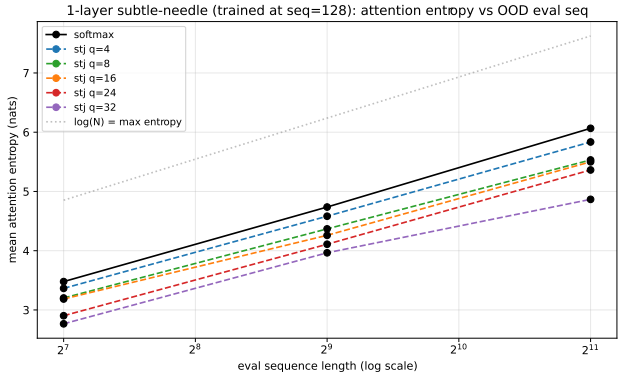


Figure 7: Mean attention entropy versus evaluation sequence length (log scale) for softmax and Stieltjes at $q \in \{4, 8, 16, 24, 32\}$, one-layer subtle-needle, $N_{\text{train}} = 128$. All variants grow roughly as $\log(N)$ (dotted line), consistent with attention spreading over more tokens at longer sequences. The vertical offset decreases with q: higher q maintains lower entropy throughout, quantifying the slower dispersion of polynomial versus exponential tails.

Taken together, these results describe the operating envelope of Stieltjes attention. At one-layer scale, $q \in [2, 8]$ achieves near-softmax accuracy (Table 3); within the conservative $q \in [2, 4]$ sub-band, Stieltjes tolerates $64\times$ length extrapolation on this task family. Beyond $q = 16$ the one-layer subtle-needle accuracy drops sharply (0.254 at $q = 24$, 0.091 at $q = 32$), reflecting a harder optimization landscape at very high concentration. At six-layer scale, in-distribution accuracy at $q \in \{8, 16\}$ stays within ≤ 0.03 of the softmax baseline on every CLRS task tested (Table 2). Length extrapolation at depth (Table 4) is a separate and more demanding regime: even at $q = 4$, six-layer models trained at $N_{\text{train}} = 2048$ show an attention-output standard deviation decay of 35–50 % at $8\times$ evaluation length, concentrated in the deeper layers; whether $q \in \{8, 16\}$ models extrapolate better than $q = 4$ at this depth is a direction for future work. The implementation consequence is that paper-shipped Stieltjes models should report their training q and N_{train} as part of the model definition, alongside the iteration count T of Section 4.4.

At a stricter capacity constraint—*one-head, one-layer* with $d_{\text{embd}} = 64$ and NoPE at $N_{\text{train}} = 128$—the q-dependence develops a complementary low-end failure mode. On the pointing task *max* (predict the single largest input value), two-seed runs show that $q \in \{4, 8\}$ and softmax all extrapolate to `accuracy_fixed` $= 1.000$ out to $128 \times N_{\text{train}} = 16{,}384$, while $q = 2$ alone collapses: the seed-$\{1, 42\}$ mean drops from 1.000 at $4\times$ to 0.199 at $128\times$, with both seeds collapsing in the same direction (magnitude differs; seed 42 collapses earlier). We interpret this as a peak-sharpness failure: at $q = 2$ the softer polynomial peak cannot isolate a single correct key among many candidates once the context grows long, whereas $q \geq 4$ sharpens sufficiently to retain the pointing behaviour. The high-q end of the same capacity regime is training-unstable rather than systematically worse: on subtle-needle at $q = 64$, a seed-42 run achieves near-perfect extrapolation out to $16{,}384$ but the seed-1 replication fails to converge at all (`accuracy_fixed` ≤ 0.002 at every evaluated length). We do not claim $q = 64$ as a robust configuration; we include the observation to calibrate expectations for extreme q values at one-head scale. Full two-seed tables for both tasks are in Appendix B.

5 Discussion

5.1 Stieltjes as a narrow-band alternative to softmax for length extrapolation

The empirical picture from Sections 4.3 and 4.5 has two axes. In-distribution at the six-layer scale on $N = 128$ CLRS tasks, the constant $\lambda_0 = 1.1$ initialization supports a wide usable band: Stieltjes at $q \in \{8, 16\}$ matches softmax within ≤ 0.03 on every tested task (Table 2), so *in-distribution* Stieltjes is not narrow-band. It is narrow-band for *length extrapolation*: at one-layer scale on subtle-needle, $q \in [2, 4]$ tolerates $64\times$ extrapolation while $q = 8$ and $q = 16$ hit a lower, constant ceiling (Table 3); at six-layer scale on $N_{\text{train}} = 2048$ needle, even $q = 4$ begins to degrade by $2\times$ (Table 4). The useful operating envelope is therefore a product of the two axes: for in-distribution training, q up to 16 is

available; for length-extrapolation-at-depth, $q \in [2, 4]$ remains the conservative recommendation pending an extrapolation sweep at depth.

The transition in the length-extrapolation axis is sharp rather than gradual: at one-layer scale $q = 4$ extrapolates to $64\times$, $q = 8$ is volatile at $16\times$, $q \geq 16$ is volatile already at $4\times$. A complementary low-q wall appears at tight capacity: on the *max* pointing task at one-head, one-layer, $q = 2$ alone collapses at extrapolation (`accuracy_fixed` mean dropping from 1.000 at $4\times$ to 0.199 at $128\times$) while $q \in \{4, 8\}$ and softmax all retain 1.000 (Section 4.5). The usable band is thus bounded on both sides, and any adaptive or learnable-q scheme for extrapolation-heavy workloads must respect both walls; in-distribution training at higher q is nevertheless available where length extrapolation is not required.

5.2 Multi-pass tradeoff

The fundamental cost of Stieltjes attention relative to softmax is the number of HBM sweeps per query block. Standard flash attention requires essentially one pass over the key/value blocks; Stieltjes attention requires $(2+T)$ passes: one for the row maximum, T for Newton–Raphson (each evaluating $f(\lambda)$ and $f'(\lambda)$), and one for output accumulation. At $T = 3$, this is $5\times$ the HBM traffic per query block versus flash softmax.

Despite this, our Triton kernel still beats the flash-style PyTorch Stieltjes reference substantially on forward latency and peak memory across both hardware targets (concrete numbers reported only in Section 4.2). The speedup comes from three sources: each pass performs simple arithmetic (no transcendental functions for $q = 1$, and only one $\exp(-q \log(\cdot))$ pair per element for general q); the query block remains in SRAM across all passes, so only key/value tiles stream from HBM; and Triton's autotuning selects tile sizes that maximize occupancy. The peak speedup is higher on H100 than on A100, consistent with our tile-based implementation exploiting the newer tensor-core throughput more aggressively.

We make no claim that the multi-pass kernel will outperform a fused flash softmax kernel at matched precision. It does, however, beat the reference path that any user of Stieltjes attention would otherwise be stuck with, by a factor that makes training-time deployment practical.

5.3 NR iteration count as part of the model definition

The train-eval NR-count coupling (Section 3.2) is novel for algebraic attention. Its mechanism is simple: because the training initialization $\lambda_i^{(0)} = 1.1$ is applied to score-centered rows rather than to the true root of F, the converged λ at any finite T depends on the learned score distribution, and a fixed T stops short of the true root by a data-dependent residual. Models absorb this residual into their learned QKV projections at training time; changing T at inference presents them with a different residual that their weights do not compensate for.

The practical consequence is that T is part of the model definition, much as the positional encoding family is. A Stieltjes model shipped without its training T is not fully specified. For reproducibility, we recommend logging T alongside other attention hyperparameters and treating it as a build-time parameter rather than a runtime toggle.

5.4 Future work

1. Widening the q operating band: curriculum schedules that anneal q more gradually, or architectural modifications (e.g. learnable per-head q) that avoid the sharp $q \approx 4 \to 8$ transition documented here.

2. Testing Stieltjes on a real language modeling benchmark (perplexity on WikiText or C4). Our characterization here is algorithmic-reasoning-focused and synthetic.

3. Extending the kernel to FP8 precision and to longer sequences through improved occupancy and pipelining, to bring the multi-pass overhead further down at very long context.

6 Conclusion

We have presented a flash-style Triton implementation of *Stieltjes attention*, an algebraic replacement for softmax whose attention weights take the form $p_j = (\lambda - s_j)^{-q}$ with λ determined by per-row Newton–Raphson root finding. The forward kernel attains a substantial speedup and memory reduction on A100 and H100 against a flash-style PyTorch Stieltjes reference (Section 4.2). At moderate model scale and sequence length, Stieltjes with $q \in \{2, 4, 8, 16\}$ under the constant $\lambda_0 = 1.1$ initialization matches softmax within ≤ 0.03 on a suite of algorithmic-reasoning tasks (Table 2); and with $q \in [2, 4]$ it extrapolates robustly to $64\times$ the training sequence length on subtle-needle retrieval at one-layer scale with NoPE. Two kernel-side caveats bound the forward-kernel contribution: the Triton backward is numerically unstable on trained attention patterns, so all accuracy-producing training in this paper uses the PyTorch-reference Stieltjes path (Section 3.3.2); and the Triton forward itself exhibits an accuracy dip on intermediate sequence lengths $N \in \{2048, 4096\}$ that we have not yet root-caused, limiting our accuracy measurements to the reference path. Both are future work.

We also document the Newton–Raphson iteration count T as a training hyperparameter: the trained weights absorb the finite-T residual and therefore expect the same T at inference. This makes T part of the model definition alongside positional-encoding and sparsity-pattern choices.

Our full implementation, training harness, benchmark scripts, and reproducibility data are released at `https://github.com/gulkoa/stieltjes-triton`. Future work will characterize Stieltjes on real language-modeling benchmarks, investigate learnable per-head q to widen the operating band, and extend the kernel to FP8 precision for production deployment.

References

Mathieu Blondel, André F. T. Martins, and Vlad Niculae. Learning with Fenchel–Young losses. *Journal of Machine Learning Research*, 21(35):1–69, 2020.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations (ICLR)*, 2021. arXiv preprint arXiv:2009.14794.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

Gonçalo M. Correia, Vlad Niculae, and André F. T. Martins. Adaptively sparse transformers. *arXiv preprint arXiv:1909.00015*, 2019.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv preprint arXiv:2307.08691, 2023.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *International Conference on Machine Learning (ICML)*, 2021. arXiv preprint arXiv:2103.03404.

Nuno Gonçalves, Marcos Treviso, and André F. T. Martins. AdaSplash: Adaptive sparse flash attention. In *International Conference on Machine Learning (ICML)*, 2025. arXiv preprint arXiv:2502.12082.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning (ICML)*, pages 5156–5165, 2020.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

André F. T. Martins and Ramón Fernandez Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International Conference on Machine Learning (ICML)*, pages 1614–1623, 2016.

Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*, 2018.

Ken Nakanishi. Scalable-softmax is superior for attention. *arXiv preprint arXiv:2501.19399*, 2025.

Vlad Niculae and Mathieu Blondel. A regularized framework for sparse and structured neural attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. arXiv preprint arXiv:1705.07704.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

Ohio Supercomputer Center. Ohio supercomputer center, 1987. URL `https://ror.org/01apna436`.

Jeffrey Pennington and Pratik Worah. Nonlinear random matrix theory for deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. arXiv preprint arXiv:1710.10088.

Ben Peters, Vlad Niculae, and André F. T. Martins. Sparse sequence-to-sequence models. In *Association for Computational Linguistics (ACL)*, 2019.

Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. cosFormer: Rethinking softmax in attention. In *International Conference on Learning Representations (ICLR)*, 2022. arXiv preprint arXiv:2202.08791.

Jason Ramapuram, Federico Danieli, Eeshan Dhekane, Floris Weers, Dan Busbridge, Pierre Ablin, Tatiana Likhomanenko, Jagrit Digani, Zijin Gu, Amitis Shidani, and Russ Webb. Theory, analysis, and best practices for sigmoid self-attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv preprint arXiv:2409.04431.

Hemanth Saratchandran, Jianqiao Zheng, Yiping Ji, Wenbo Zhang, and Simon Lucey. Rethinking attention: Polynomial alternatives to softmax in transformers. *arXiv preprint arXiv:2410.18613*, 2024.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision. *arXiv preprint arXiv:2407.08691*, 2024.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6), 2022. arXiv preprint arXiv:2009.06732.

Philippe Tillet, H. T. Kung, and David Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *Workshop on Machine Learning and Systems (MLSys)*, 2019.

Constantino Tsallis. Possible generalization of Boltzmann–Gibbs statistics. *Journal of Statistical Physics*, 52(1–2):479–487, 1988.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

Petar Veličković, Christos Perivolaropoulos, Federico Barbero, and Razvan Pascanu. Softmax is not enough (for sharp size generalisation). In *International Conference on Machine Learning (ICML)*, 2025. arXiv preprint arXiv:2410.01104.

Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

Mitchell Wortsman, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Replacing softmax with ReLU in vision transformers. *arXiv preprint arXiv:2309.08586*, 2023.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv preprint arXiv:2309.17453.

Tianzhu Ye, Li Dong, Yuqing Xia, Yutao Sun, Yi Zhu, Gao Huang, and Furu Wei. Differential transformer. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv preprint arXiv:2410.05258.

Zayd Muhammad Kawakibi Zuhri, Imam Nur Bani Fuadi, et al. Softpick: No attention sink, no massive activations with rectified softmax. *arXiv preprint arXiv:2504.20966*, 2025.

A Implementation details

All kernels are implemented in Triton [Tillet et al., 2019] and wrapped in `torch.autograd.Function` for PyTorch integration. This appendix collects the low-level choices that support the numerical-stability and throughput summaries in Section 3.3.

A.1 Autotuning

The standalone Stieltjes kernel (row-wise normalization without the full attention computation) uses Triton's `@triton.autotune` over five configurations: block sizes $\{256, 512, 1024, 2048, 4096\}$ with warp counts $\{2, 4, 4, 8, 8\}$ and 2 pipeline stages, keyed on `n_cols`. The flash-attention kernel uses fixed block sizes BLOCK_M = BLOCK_N = 64 (or 128×64 for head dimension $D \leq 64$) to match the tile shapes of the standard Triton flash-attention tutorial, facilitating a direct comparison.

A.2 Numerical stability

Section 3.3 describes the three stability devices used in the forward kernel; this subsection records the numerical constants for reproducibility. The difference clamp is $\epsilon = 10^{-6}$ in the flash kernel and $\epsilon = 10^{-9}$ in the standalone (row-wise) kernel; the causal sentinel is -10^{30} (a representable single-precision value safely below any score encountered in training).

A.3 $q = 1$ fast path

The compile-time branch (`if sq == 1.0` in Triton JIT) replaces `tl.log` and `tl.exp` with a single reciprocal and a squaring operation, yielding a $\sim 15\%$ throughput improvement in microbenchmarks.

A.4 Saved tensors for the backward pass

The forward kernel stores two `float32` vectors per query row for the backward pass:

- The un-centered pole $\lambda_i + r_{\max,i}$, so that the backward kernel can recompute weights directly from raw scores without re-centering.
- $D_i = \sum_{j \leq i} (\lambda_i - s_{ij})^{-q-1}$, the sum of derivative weights used in the δ_i correction.

These two N-element vectors per head are the *only* auxiliary storage beyond $\mathbf{Q}, \mathbf{K}, \mathbf{V}$, the analog of the row max m_i and log-sum-exp ℓ_i saved by FlashAttention.

A.5 Causal masking

Causal masking reuses FlashAttention's convention: masked positions in $\mathbf{K}$ are loaded as the sentinel -10^{30}, so that after centering $\tilde{s}_{ij} \approx -10^{30}$ and $\Delta_{ij} = \lambda_i - \tilde{s}_{ij} \approx 10^{30}$, giving $\Delta_{ij}^{-q} \approx 0$. No explicit branch is needed in the inner loop; the masking falls out algebraically. In the backward recompute, gradient tiles $\mathrm{d}\mathbf{S}$ are explicitly zeroed at masked positions to prevent gradient leakage.

B One-head one-layer characterization

To expose q-dependent behaviour below the multi-head capacity of Table 2, we train every CLRS task at a deliberately under-capacity configuration: *one-head, one-layer* with $d_{\text{embd}} = 64$, NoPE, $N_{\text{train}} = 128$, otherwise identical recipe (50 epochs, 20,000 training / 2,000 validation samples, constant $\lambda_0 = 1.1$, $T = 3$). Two seeds per configuration; evaluation is `accuracy_fixed` on the extrapolation sweep $N \in \{128, 512, 2048, 8192, 16384\}$, scaled `max-arr-len` $= N - 8$. This probe amplifies differences in the attention-normalization kernel because no additional heads are available to absorb them. The one-head *max* sweep yields the paper's main capacity-gated claim (Section 4.5) and is tabulated in full below, followed by the *subtle_needle* sweep.

B.1 Max: $q = 2$ extrapolation collapse (two-seed robust)

seed	attn	$N = 128$	512	2048	8192	16384
42	softmax	0.510	1.000	1.000	1.000	1.000
42	$q = 2$	0.519	1.000	0.643	0.146	0.073
42	$q = 4$	0.519	1.000	1.000	1.000	1.000
42	$q = 8$	0.513	1.000	1.000	1.000	1.000
1	softmax	0.516	1.000	1.000	1.000	1.000
1	$q = 2$	0.512	1.000	1.000	0.690	0.325
1	$q = 4$	0.516	1.000	1.000	1.000	1.000
1	$q = 8$	0.516	1.000	1.000	1.000	1.000

Table 5: *max* one-head one-layer two-seed extrapolation sweep, `accuracy_fixed` on 2,000 validation samples per length. $q = 2$ collapses monotonically on both seeds; $q \in \{4, 8\}$ and softmax extrapolate to 1.000. The training length $N = 128$ reads near 0.5 uniformly because the scaled `max-arr-len` at $N = 128$ produces near-deterministic tie structure between the top candidates; longer evaluations deterministically pin the argmax.

Two-seed evidence that Stieltjes at $q = 2$ fails the pointing task at extrapolation while $q \in \{4, 8\}$ and softmax all succeed. The mechanistic interpretation—polynomial peak too soft to isolate a single key among many candidates—is summarized in Section 4.5.

B.2 Subtle-needle: parity absent, high-q training seed-fragile

seed	attn	$N = 128$	512	2048	8192	16384
42	softmax	0.917	0.666	0.266	0.206	0.223
42	$q = 2$	0.766	0.206	0.035	0.022	0.016
42	$q = 4$	0.788	0.384	0.062	0.022	0.010
42	$q = 8$	0.817	0.524	0.172	0.058	0.037
42	$q = 16$	0.323	0.000	0.000	0.000	0.000
42	$q = 24$	0.631	0.000	0.000	0.000	0.000
42	$q = 32$	0.444	0.000	0.000	0.000	0.000
42	$q = 64$	0.510	0.949	1.000	1.000	1.000
1	softmax	0.922	0.694	0.268	0.209	0.191
1	$q = 2$	0.764	0.160	0.024	0.013	0.019
1	$q = 4$	0.810	0.337	0.052	0.021	0.023
1	$q = 8$	0.827	0.516	0.140	0.062	0.037
1	$q = 16$	0.925	0.622	0.172	0.093	0.058
1	$q = 24$	0.639	0.498	0.269	0.184	0.159
1	$q = 32$	0.507	0.510	0.373	0.278	0.243
1	$q = 64$	0.000	0.002	0.000	0.000	0.000

Table 6: *subtle_needle* one-head one-layer two-seed sweep, `accuracy_fixed`. Softmax outperforms every Stieltjes configuration on both seeds; $q = 8$ is the best Stieltjes choice but still trails softmax. The high-q rows exhibit *opposite* seed patterns: seed-42 fails at $q \in \{16, 24, 32\}$ but extrapolates perfectly at $q = 64$; seed-1 learns moderately at $q \in \{16, 24, 32\}$ but fails entirely at $q = 64$.

At one-head no Stieltjes configuration matches softmax on *subtle_needle*, consistent with the main-table qualitative ordering but at a smaller absolute scale. The high-q regime ($q \geq 16$) is training-

unstable: which configuration converges and which fails is an initialization question, not a kernel question. Section 4.5 states this succinctly and we include the complete table here for transparency.